Dynamical systems over Galois fields: Applications to DES and to the Signal Language

Michel Le Borgne

Dynamical systems over Galois fields: Applications to DES and to the SIGNAL Language

Michel Le Borgne
IRISA/INRIA, Campus de Beaulieu
35042 RENNES CEDEX, FRANCE
e-mail : leborgne@irisa.fr

April 7, 1993

1 Introduction

Polynomial dynamical systems over Galois fields $\mathcal{F}_p = Z/pZ$ where p is a prime, were introduced for studying the logical and synchronisation properties of SIGNAL programs (with $p = 3$). It appeared rapidly that these aspects of SIGNAL programs were very closed to Discrete Event Systems (DES). It is always possible to code a DES (at least those specified by finite state automata), as a dynamical system over some Galois field. The use of polynomial functions over finite fields provides us with efficient algorithms to represent these functions and polynomial equations. These methods have been discovered by people working in the field of circuit verification and were established for boolean functions i.e. polynomial functions over $Z/2Z$. The graph data structure which is at the heart of the method generalizes quite easily to the other fields Z/pZ. So it was natural to try to use dynamical systems over Galois fields to solve various control problems in DEDS theory. With these techniques we hope to master, to some extent, the combinatoric explosion experienced with automata composition: the number of states of the resulting automata is the product of the numbers of state of each automata. In contrast, the composition of two dynamical systems is simply obtained by putting the equations together, as usual in mathematics when combinaisons of equations are considered.

Two problems have to be solved for a wide application of logical DES theory: the description of large plants and the specification of properties. We need languages to specify both. The SIGNAL language, despite it was designed for the specification of hybrid systems, has some interesting features which may be included in a language for specifying plants. Specifying properties for verification or for synthesis of supervisors is also a difficult problem. We will restrict in this paper to some general properties of reactive systems.

The paper is organized as follows: in section II the theory of dynamical systems over Galois fields is presented. Section III is devoted to the algorithmic aspects. SIGNAL is briefly presented in section IV. Section V is devoted to the proof of properties of SIGNAL programs while control problems are adressed in section VI.

2 Polynomial dynamical systems: basic properties

2.1 General form of polynomial dynamical systems

Dynamical systems over Galois fields are defined by three sets of equations:

$$\begin{aligned} Q(X,Y) &= 0 \\ X' &= P(X,Y) \\ Q_0(X) &= 0 \end{aligned}$$

where X is the vector of *state variables* (or *states* for short) and Y the vector of *events*. The first set of equations denoted $Q(X,Y)=0$ is called the *constraint equation*, it specifies which event may occur in a given state. The second equation is the *state transition equation* where X' denotes the next value of the state. The last equation put some constraints on the initial state.

In the sequel, we will use the notations $\mathcal{F}_p[X]$, $\mathcal{F}_p[X,Y]$ for the rings of polynomials in the variables X and X,Y, with coefficients in $\mathcal{F}_p$. The dimensions of X and Y will be denoted respectively n and m and their elements respectively $X_1,\ldots,X_n$ and $Y_1,\ldots,Y_m$. A dynamical system will be denoted (P,Q).

Each element (x,y) in $\mathcal{F}_p^{n+m}$ such that $Q(x,y)=0$ is termed *admissible*, an event y is admissible in the state x if (x,y) is admissible. A *trajectory* $(x_i,y_i)_{i\geq 0}$ initialized in (x_0,y_0) is a sequence of admissible pairs (x_i,y_i) such that $x_{i+1}=P(x_i,y_i)$. We will also consider state trajectories and event trajectories. The orbit of the dynamical system is the set of states x such that x belongs to a state trajectory initiated in the set of initial states defined by the equation $Q_0(X)=0$.

2.2 Ideals, morphisms, and basic tools

The theory of polynomial dynamical systems [5] uses classical tools in elementary algebraic geometry: varieties, ideals, and morphisms. This theory allows us to translate properties of sets into equivalent properties of associated ideals of polynomials.

Ideals and algebraic sets associated with systems of equations. With the equation $Q(X,Y)=0$, we associate the ideal $<Q>$ spanned by the collection of polynomials $Q(X,Y)$ and the algebraic set $\mathcal{V}(<Q>)$ of all (x,y) such that $Q(x,y)=0$.

Ideals associated with sets. On the other hand, given a subset S of $\mathcal{F}_p^{n+m}$, the set of polynomials R such that $R(s)=0$ for all $s\in S$, is an ideal of $\mathcal{F}_p[X,Y]$ we denote by $\mathcal{I}(S)$.

Some basic properties. Firstly, it holds that :

$$\mathcal{I}(\mathcal{V}(<Q>)) = <Q, X^p-X, Y^p-Y>$$

where X^p-X (resp. Y^p-Y) stands for the set of polynomials $X_i^p-X_i$ (resp $Y_i^p-Y_i$). This equality establishes a one-to-one and onto relation between subsets of $\mathcal{F}_p^{n+m}$ and *completed* ideals, i.e., ideals containing the polynomials X^p-X, Y^p-Y. From now we will only consider completed ideals and $<Q>$ will denote $<Q, X^p-X, Y^p-Y>$.

If $\underline{a}$ is such a completed ideal, the following property is easily verified :

$$P \in \underline{a} \ \ \& \ \ Q \in \underline{a} \Leftrightarrow \left(P^{p-1} + Q^{p-1}\right)^{p-1} \in \underline{a}$$

As a consequence of the above property, *every completed ideal is spanned by a single generator called a principal generator.* These particular generators are used in the practical implementation of the algorithms on ideals.

The one-to-one and onto correspondance between completed ideals and sets allows us to work not with polynomials but with elements of the quotient ring $\mathcal{F}_p[X,Y]/ < X^p - X, Y^p - Y >$, that is with polynomial functions. This remark is essential for the implementation of the algorithms presented further. For sake of notational simplicity we will continue to work with polynomials in this paper. A set Q of polynomial will be considered as a set of generators of a completed ideal $\underline{a}$ if none of the corresponding polynomial functions is equal to zero and the set of functions generates the ideal $(\underline{a} + < X^p - X, Y^p - Y >)/ < X^p - X, Y^p - Y >$.

Notice also that any decreasing sequence of completed ideals is stationnary.

Morphisms. A *morphism* is a polynomial map $P : \mathcal{F}_p^{n+m} \rightarrow \mathcal{F}_p^n$. In fact, due to the finitness of the field $\mathcal{F}_p$, every map $f : \mathcal{F}_p^{n+m} \rightarrow \mathcal{F}_p^n$ is a morphism. With the morphism P is associated a *comorphism* P^* from $\mathcal{F}_p[X]$ into $\mathcal{F}_p[X,Y]$ defined by: $p \in \mathcal{F}_p[X] : P^*(p(X)) = P^*(p(X_1, \ldots, X_n)) = p(P_1(X,Y), \ldots, P_n(X,Y))$, where $P_1, \ldots, P_n$ are the components of P. Informally, $P^*(p)$ is obtained by substituting every X_i in p with the corresponding P_i.

The following relations are straightforward:

$$\mathcal{I}(P(\mathcal{V}(\underline{a}))) = P^{*-1}(\underline{a}) \quad (1)$$

$$\mathcal{V}(< P^*(\underline{b}) >) = P^{-1}(\mathcal{V}(\underline{b})) \quad (2)$$

for all completed ideals $\underline{a}$ in $\mathcal{F}_p[X,Y]$ and all ideal $\underline{b}$ in $\mathcal{F}_p[X]$.

2.3 Properties of Polynomial Dynamical Systems

2.3.1 Liveness

Recall a system is alive if, roughly speaking, no deadlock can occur. This definition is formalized as follows.

Definition 1 *A state x is alive if there exists a signal y such that $Q(x,y) = 0$; a set of states V is alive if every element in V is alive.*

A dynamical system is alive if for all (x,y) such that $Q(x,y) = 0$, $P(x,y)$ is an alive state.

To check if V is alive, we only have to test if $V \subseteq proj_X(\mathcal{V}(< Q >))$ or, equivalently, if $< Q > \cap \mathcal{F}_p[X] \subseteq \mathcal{I}(V)$. Equivalently, a system is alive iff $P(\mathcal{V}(< Q >)) \subseteq proj_X(\mathcal{V}(< Q >))$. Using (1), these properties can be translated into equivalent ones for ideals as follows :

Proposition 1 *A dynamical system is alive if and only if*

$$P^*(< Q > \cap \mathcal{F}_p[X]) \ \subseteq \ < Q > .$$

2.3.2 Invariance

Definition 2 *A subset E of states is invariant for a dynamical system if for every x in E and every y admissible in the state x, the state $x' = P(x, y)$ is in E.*

We have the following property:

Proposition 2 *A subset E is invariant if and only if:*

$$P^*(\mathcal{I}(E)) \subseteq < Q > + \mathcal{I}(E)\mathcal{F}_p[X, Y],$$

where $\mathcal{I}(E)\mathcal{F}_p[X, Y]$ is the ideal generated by $\mathcal{I}(E)$ in $\mathcal{F}_p[X, Y]$.

The proof is straightforward and can be found in [6].

This kind of property covers the class of so-called "safety" properties i.e., properties that intuitively state that "nothing bad can happen" or, in our terminology, that "the set of good states is invariant". It may happen that the set of "good states" V is not invariant. In this case, it is interesting to find the *largest invariant subset* of V.

Notice that $\mathcal{F}_p[X, Y] \cong (\mathcal{F}_p[X])[Y]$, hence a polynomial $p \in \mathcal{F}_p[X, Y]$ may be rewritten as a polynomial in the variables Y with polynomials in X as coefficients. So, for $p \in \mathcal{F}_p[X, Y]$, denote by

$$\mathrm{Coef}_X(p) \tag{3}$$

the ideal of $\mathcal{F}_p[X]$ spanned by its coefficients when rewritten as a polynomial in the variables Y. $\mathrm{coef}_X(\underline{a})$ is defined accordingly for any completed ideal $\underline{a} \subseteq \mathcal{F}_p[X, Y]$. Given any of generators $\{\alpha_1(X, Y), \ldots, \alpha_k(X, Y)\}$ of $\underline{a}$ (as defined previously) $\mathrm{coef}_X(\underline{a})$ is generated by

$$\{\mathrm{coef}_X(\alpha_1), \ldots, \mathrm{coef}_X(\alpha_k)\}$$

The geometric meaning of $coef_X$ is captured by the following relation:

$$\mathcal{V}(\mathrm{coef}_X(\underline{a})) = \{x \ : \ \forall y, (x, y) \in \mathcal{V}(\underline{a})\} \tag{4}$$

With this definition we get the straightforward proposition:

Proposition 3 *Given an ideal $\underline{a}$ in $\mathcal{F}_p[X]$ and an ideal $\underline{b}$ in $\mathcal{F}_p[X, Y]$, the following assertions are equivalent:*

i) $\underline{b} \subseteq \underline{a}\mathcal{F}_p[X, Y]$

ii) $coef_X(\underline{b}) \subseteq \underline{a}$

Using this property, we get an algorithm to compute the largest invariant subset included in a given set E of states. Consider the following chain of ideals:

$$\begin{aligned}
\underline{a}_0 &= \mathcal{I}(E) \\
\underline{a}_1 &= \underline{a}_0 + coef_X(P^*(\underline{a}_0) \cap < Q >^c) \\
&\ldots \\
\underline{a}_k &= \underline{a}_{k-1} + coef_X(P^*(\underline{a}_{k-1}) \cap < Q >^c)
\end{aligned}$$

where $\underline{a}^c$ denotes the ideal associated with the complementary set of $\mathcal{V}(\underline{a})$. Any (increasing) chain of ideals must be stationary. Hence, let $\underline{a}(E, P)$ denote the largest ideal of this chain. The proof of the following result is straightforward :

Theorem 1 *With the previous notations, if* $\underline{a}(E,P)$ *is not the entire ring, then* $\mathcal{V}(\underline{a}(E,P))$ *is the largest invariant subset of* E*. If* $\underline{a}(E,P)$ *is the entire ring* $\mathcal{F}_p[X]$ *then* E *contains no proper invariant subset.*

The absence of invariant subsets is also an interesting property. It means that every trajectory entering E *may* eventually leave E. Thus *fairness* properties can be viewed as particular instances of invariance properties.

An important application of the notion of largest invariant subset is to determine whether a set of states V can be reached starting from a state x_0. This problem is equivalent to that of testing if x_0 belongs to the largest invariant subset U of $\mathcal{F}_p^n - V$, the complement of V in the state space. If x_0 belongs to U, then every trajectory starting from x_0 stays in U, so V cannot be reached from x_0. Conversely, if x_0 does not belong to U, then there is a trajectory intialized in x_0 that leaves $\mathcal{F}_p^n - V$, i.e., this trajectory reaches V.

2.3.3 Control-invariance

An other useful property is that of *control-invariance* of a set of states E:

Definition 3 [1] E *is control-invariant if for all state* x *in* E*, there exists an event* y *such that* $Q(x,y) = 0$ *and* $P(x,y)$ *is still an element of* E*.*

Whereas the invariance property of a set E states that every trajectory initialized in E never leaves E, the above property means that it is possible starting from E to allways stay in E. Equivalently, the system is E-stabilizable if we can modify the system, by adding new contraints, in such a way that E becomes invariant.

Proposition 4 E *is control-invariant if and only if:*

$$(< Q > + P^*(\mathcal{I}(E))) \cap \mathcal{F}_p[X] \ \subseteq \ \mathcal{I}(E)$$

As for the largest invariant subset of a given set E, it is possible to compute the largest control-invariant subset of E. The algorithm is as follows:

$$\begin{aligned} \underline{a}_0 &= \mathcal{I}(E) \\ \underline{a}_1 &= \underline{a}_0 + (\underline{c} + P^*(\underline{a}_0)) \cap \mathcal{F}_p[X] \\ &\cdots \\ \underline{a}_{k+1} &= \underline{a}_k + (\underline{c} + P^*(\underline{a}_k)) \cap \mathcal{F}_p[X] \end{aligned}$$

where $\underline{c} =< Q >$. The proof of correctness is similar to the proof of the largest invariant algorithm. It is also of interest to check whether a set of states E does not contain any control-invariant subset. This means that all trajectory entering E *must* exit E after a finite delay. This kind of property may be used for checking the absence of *starvation*.

[1] Our definition of control-invariance is stronger than the usual one. It implies some kind of liveness. There is no difference with the usual definition if we restrict to alive states

2.3.4 Derived properties

Many properties may be derived from the three basic properties. For example, a set of states E is attractive if and only if every state trajectory reaches E. It is easy to prove that E is attractive if and only if the initial states are not in the greatest control-invariant subset of $\mathcal{F}_p^n - E$. In the same way, if we define a recurrent set E as a set which is infinitely often visited by any trajectory, it is easy to show that a set E is recurrent if and only if $O - E$ doesn't contain any control-invariant subset, where O is the orbit of the dynamical system.

3 Algorithmic issues

In the preceeding section a set of algorithms has been presented to solve problems for dynamical systems over Galois fields. In this section we present an efficient technique for implementing polynomial functions. The first use of this technique was for $p = 2$ (boolean functions). It generalizes quite easily to other values of p. For the sake of simplicity, we present the case $p = 3$.

It turns out that all algorithms we introduced, only use a small set of *primitive algorithms* involving polynomials or ideals. To summarise we have to:

- *Compute* $P^*(\underline{a})$, the image of ideal $\underline{a}$ by the comorphism P^* (note that $P^*(\underline{a})$ is *not* an ideal in general). This is the basic tool for analysing state transitions.
- Check if a polynomial P belongs to an ideal $\underline{a}$, and, more generally, if an ideal $\underline{a}$ is included in another one $\underline{b}$. This is equivalent to testing for inclusions of set of states or events.
- Compute intersections of the form $\underline{a} \cap \mathcal{F}_3[X]$ for some ideal $\underline{a}$ in $\mathcal{F}_3[X, Y]$. This allows us to compute projections.
- Compute the ideal associated with the complement of a set.
- Compute the coef_X of some ideal, cf. (3). This is the basis for computing largest invariant subsets.

3.1 Finitely generated algebras and families of idempotents

Given a ring $\mathcal{F}_3[X]/ < X^3 - X >$ of polynomial functions, and a variable X_i, let us define the following family of polynomial functions:

$$\begin{aligned} h^1_{X_i} &= -X_i^2 - X_i \\ h^2_{X_i} &= -X_i^2 + X_i \\ h^3_{X_i} &= 1 - X_i^2 \end{aligned} \tag{5}$$

Throughout this section, polynomials will implicitly refer to elements of $\mathcal{F}_3[X]/ < X^3 - X >$, i.e., to associated polynomial functions of our quotient ring. In other words, we shall handle polynomials as usually with the additional rule that $X^3 = X$. The three polynomial functions (5) have interesting properties:

$$(h^\alpha_{X_i})^2 = h^\alpha_{X_i} \text{ for } \alpha = 1, 2, 3 \tag{6}$$

$$h^\alpha_{X_i} h^\beta_{X_i} = 0 \text{ for all } \alpha \neq \beta \tag{7}$$

$$h^1_{X_i} + h^2_{X_i} + h^3_{X_i} = 1 \tag{8}$$

Note that, thanks to the idempotence property (6), h^α-polynomials never occur with exponent greater than 1, so *the reader should not confuse* h^1, h^2, h^3 *with exponents of* h, these superscripts can only refer to indices.

Proposition 5 *If* $P(X)$ *is an element of the quotient ring*

$$\mathcal{F}_3[X_1, \ldots, X_n]/ < X_1^3 - X_1, \ldots, X_n^3 - X_n >$$

then there exist a unique triple (P_1, P_2, P_3) *of elements of the quotient ring:*

$$\mathcal{F}_3[X_2, \ldots, X_n]/ < X_2^3 - X_2, \ldots, X_n^3 - X_n >$$

with $n-1$ *variables* $X_2, \ldots, X_n$ *such that:*

$$P(X) = h^1_{X_1} P_1 \ + h^1_{X_2} P_2 \ + h^1_{X_3} P_3$$

This theorem is a generalisation to a three valued logic of the well-known decomposition theorem of Shannon for boolean functions. From an algebraic point of view, the families $\{h^1_{X_i}, h^2_{X_i}, h^3_{X_i}\}$ are *families of orthogonal idempotents,* and the following corollary establishes that $\mathcal{F}_3[X_1, \ldots, X_n]/ < X_1^3 - X_1, \ldots, X_n^3 - X_n >$ is finitely generated by families of orthogonal idempotents.

Corollary 1 *Every element of* $\mathcal{F}_3[X_1, \ldots, X_n]/ < X_1^3 - X_1, \ldots, X_n^3 - X_n >$ *has a unique decomposition:*

$$P(X) = \sum \ a_\alpha h^{\alpha_1}_{X_1} \ldots h^{\alpha_n}_{X_n}$$

where the α_i*'s range over the set* $\{1, 2.3\}$, α *is the word* $\alpha = \alpha_1 \ldots \alpha_n$, *and* $a_\alpha \in \mathcal{F}_3$.

The proof is straightforward and relies on a recursion over the number of variables.

The decomposition of the polynomial functions using the basis of monomials $h^{\alpha_1}_{X_1} \ldots h^{\alpha_n}_{X_n}$ is not interesting for an efficient implementation of the arithmetic of polynomial functions since it needs 3^n coefficients. It is possible to have a more compact representation but at the price of chosing an order on the variables. Moreover the efficiency of the representation will depend on the chosen order.

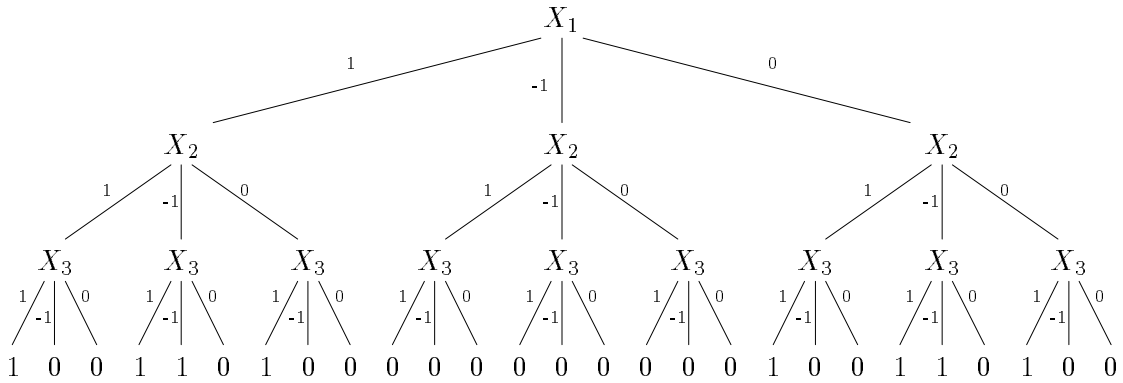
Definition 4 *Given a polynomial function* P *in* $\mathcal{F}_3[X]/ < X^3 - X >$, *and an order* $X_1 \prec X_2 \prec \ldots \prec X_n$ *on the variables, a h-expression of* P *is:*

- *either* $P(X) = c_1 h^1_{X_i} \ + c_2 h^2_{X_i} \ + \ c_3 h^3_{X_i}$ *where* $c_i \in \mathcal{F}_3$
- *or* $P(X) = h^1_{X_i} P_1 \ + h^2_{X_i} P_2 \ + h^3_{X_i} P_3$ *where the* P_i *are h-expressions with variables greater than* X_i.

A h-expression may be pictured as a ternary tree. For example the polynomial function:

$$\begin{aligned} & X_1^2 X_2^2 X_3^2 - X_1^2 X_2^2 X_3 - X_1^2 X_2 X_3^2 + X_1^2 X_2 X_3 - X_1^2 X_3^2 \\ & - X_1^2 X_3 - X_1 X_2^2 X_3^2 + X_1 X_2^2 X_3 + X_1 X_2 X_3^2 - X_1 X_2 X_3 + X_1 X_3^2 \\ & + X_1 X_3 + X_2^2 X_3^2 - X_2^2 X_3 - X_2 X_3^2 + X_2 X_3 - X_3^2 - X_3 \end{aligned}$$

is represented as:



where the left branch of a subtree with root X_i represents the $h^1_{X_i}$ factor, the middle branch represents the $h^2_{X_i}$ factor and the right branch the $h^3_{X_i}$ factor. The leaves of the whole tree are labelled with numerical coefficients. A polynomial function has several h-expressions, so h-expressions are not canonical representations. The decomposition introduced in Corollary 1 is a h-expression with respect to any ordering of the variables. All h-expressions have nice properties with respect to polynomial operations. Given an operation *op*, (sum, product...), and two h-expressions representing polynomials (with the same order on the variables), say :

$$\begin{aligned} P(X) &= h^1_{X_1} P_1 + h^2_{X_1} P_2 + h^3_{X_1} P_3 \\ P'(X) &= h^1_{X_1} P'_1 + h^2_{X_1} P'_2 + h^3_{X_1} P'_3 \end{aligned}$$

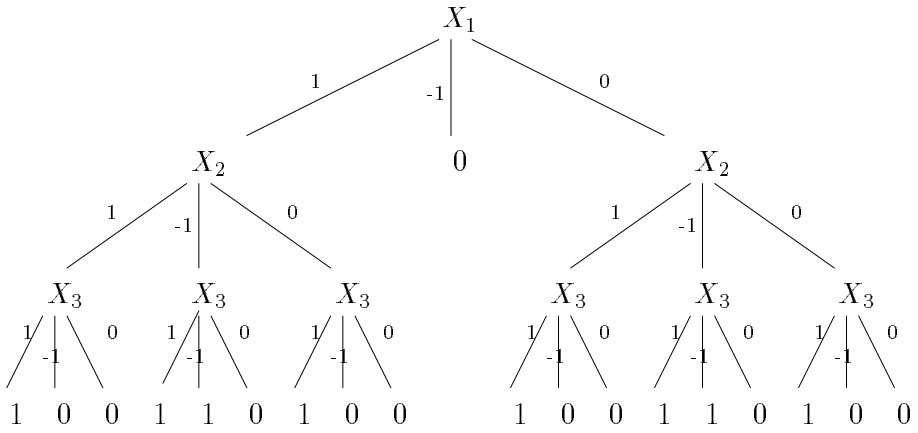
then it holds that

$$P \; op \; P' = h^1_{X_1}(P_1 \; op \; P'_1) + h^2_{X_1}(P_2 \; op \; P'_2) + h^3_{X_1}(P_3 \; op \; P'_3) \tag{9}$$

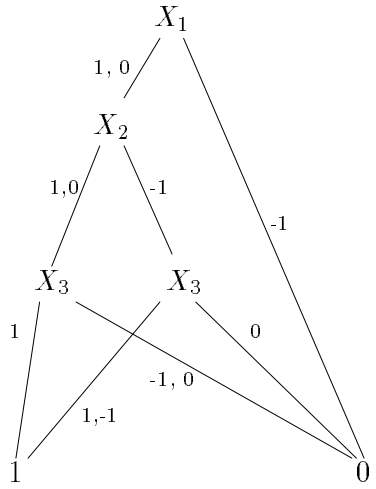
This orthogonality property would allow efficient implementations of polynomial operations if the number of monomials in such h-expressions were not, as previously noticed, too big in general for practical use.

Two ideas give the clue to an efficient implementation of polynomial functions. The first one is to reduce the h-expressions of the form $P(X) = h^1_{X_i} P_1 + h^2_{X_i} P_2 + h^3_{X_i} P_3$ by eliminating the idempotents $h^\alpha_{X_i}$ when $P_1 = P_2 = P_3$ and replacing the former expression by the common value P_1. This is possible since the sum of these idempotents is 1 and the resulting expression is still a h-expression.

The second idea comes from the fact that, in the tree representation of the h-expressions, it often happens that several subtrees are identical. Thus it is tempting to have a representation sharing the common subtrees. These two ideas lead to a generalisation of *Bryant graphs* [11, 12, 13] and fortunately these graphs retain in some way the orthogonality property (9). For example, a graph representing the polynomial function in the above example is obtained first by reducing the tree:



and then by sharing the identical subtrees:



This representation has been introduced in [11] for boolean functions. The representation given here is a slight generalisation of Bryant graphs but can be extended to all algebras generated by finite families of orthogonal idempotents. This kind of representation of polynomial functions has proved to be efficient on the average despite it is exponential in the worst case. The efficiency of the elementary operations on polynomial functions (sum, product, composition) obtained with this representation allows us to use in practice principal generators for representing ideals. We summarize now the basic toolkit of final *primitive algorithms* :

An effective toolkit of primitive algorithms :

Computing a principal generator of an ideal : if $\underline{a} =< p_1, \ldots, p_k >$ is an ideal, a principal generator is

$$g = 1 - \prod_{i=1}^{k}(1 - p_i^2) \ , \quad \text{or alternatively } p_1 \oplus \ldots \oplus p_k$$

using the notation $p \oplus q = (p^2 + q^2)^2$.

Testing for inclusion : $\underline{a}_1 \subseteq \underline{a}_2$ is equivalent to $g_2(1-g_1^2)=0$.

Computing intersections $\underline{a} \cap \mathcal{F}_3[X]$: the algorithm is based on the following simple fact: if $(X,Y)=(X_1,\ldots,X_n,Y_1,\ldots Y_m)$ then $(x,y_1,\ldots,y_{m-1})$ belongs to $proj_{X,Y_1,\ldots,Y_{m-1}}(\mathcal{V}(\underline{a}))$ if and only if there exists a y_m such that
$g(x,y_1,\ldots,y_{m-1},y_m)=0$. This is equivalent to the condition:

$$g(x,y_1,\ldots,y_{m-1},1)g(x,y_1,\ldots,y_{m-1},-1)g(x,y_1,\ldots,y_{m-1},0)=0 \qquad (10)$$

A simple iteration gives then the full projection. The actual algorithm is more subtle but basically uses this idea.

Complement of a set : if g is a principal generator of $\underline{a}$, then $1-g^2$ is a principal generator of the ideal associated with the complement of $\mathcal{I}(\underline{a})$.

Computing the $coef_X$ **of some ideal :** using (4), and noticing that $P \oplus Q = 0$ iff $P=0$ and $Q=0$, one can show that this is similar to performing (10), but with the standard product of factors replaced by the $\oplus$ operation.

Computing comorphisms : this reduces to elementary operations (sums and products) on polynomials and is performed efficiently using the graph representation.

A complete implementation of the algorithms may be founded in [7].

4 A brief description of SIGNAL

4.1 Introduction

SIGNAL is a synchronous, data-flow oriented language dedicated to real-time applications, to signal programming systems and, more generally to C^3-systems. We refer the reader to the introductory paper [1] for a thorough discussion of the principles of synchronous programming and to [2] for an extended presentation of the SIGNAL language, and we concentrate here on the minimum needed for a good understanding of this paper.

Reliable design of real time systems requires static verification of correctness, particularly for real-time programs that have to meet some critical specifications. Classical tools to reason about such non terminating, continuously interacting systems rely on temporal logic [8, 9] or automata theory [10]. An alternative approach has been choosen for SIGNAL programs. Logical and temporal aspects of programs are translated into polynomial dynamical systems, and algebraic techniques are used to prove a wide variety of properties. This approach is motivated by the equational style of SIGNAL .

4.2 The underlying model

Our model handles infinite sequences of data with a certain kind of restricted asynchronism. Assume that each sequence, in addition to the normal values it takes in its range, can also take a special value representing the *absence* of data at that instant. The symbol used for absence is $\perp$. Therefore, an infinite time sequence of data (we shall refer to informally as a *signal* in this discussion) may look like

$$1,-4,\perp,\perp,4,2,\perp,... \qquad (11)$$

which is interpreted as the signal being absent at the instants $n = 3, 4, 7, ...$ etc. A typical way of specifying such constraints will be to write equations relating different signals. The following questions are immediate from this definition:

(1) If a single signal is observed, should we distinguish the following samples from each other?

$$\{1, -4, \perp, \perp, 4, 2, \perp, ...\}\ ,\ \{\perp, 1, \perp, -4, \perp, 4, \perp, 2, \perp, ...\}\ ,\ \{1, -4, 4, 2, ...\}$$

Consider an "observer"[2] who monitors this single signal and does nothing else. Since he is assumed to observe only *present* values, there is no reason to distinguish the samples above. In fact, the symbol $\perp$ is simply a tool to specify the *relative* presence or absence of a signal, given an *environment,* i.e. other signals that are also observed. Jointly observed signals taking the value $\perp$ simultaneously for any environment will be said to *possess the same clock,* and they will be said to possess different clocks otherwise. Hence clocks may be considered as equivalence classes of signals that are present simultaneously. This notion of time makes no reference to any "physical" universal clock: time is rather local to each particular subset of signals in consideration.

(2) How to interconnect two systems? Consider the following two systems specified via equations:

$$y_n = \text{ if } x_n > 0 \text{ then } x_n \text{ else } \perp \tag{12}$$

and the usual addition on sequences, namely

$$z_n = y_n + u_n \tag{13}$$

In combining these systems, it is certainly preferable to match the successive occurrences $y_1, y_2, ...$ in (13) with the corresponding *present* occurrences in (12) so that the usual meaning of addition be met. But this is in contradiction with the bruteforce conjunction of equations (12,13)

$$\begin{aligned} y_n &= \text{ if } x_n > 0 \text{ then } x_n \text{ else } \perp \\ z_n &= y_n + u_n \end{aligned}$$

which yields $z_n = \perp + u_n$ whenever $x_n \leq 0$. To summarize, our formalism will provide a *multiform* but *coherent* notion of time. Other formalisms using the same approach to handle time are the so-called *synchronous languages* [1, 3, 4].

4.3 SIGNAL-kernel

We shall introduce only the primitives of the SIGNAL language, and drop any reference to typing, modular structure, and various declarations; the interested reader is referred to [2]. SIGNAL handles infinite sequences of data with time implicit: such sequences will be referred to as *signals.* Instructions of SIGNAL are intended to relate *clocks* as well as *values* of the various signals involved in a given system. We term a system of such relations *program*; programs may be used as modules and further combined as indicated later.

A basic principle in SIGNAL is that a single name is assigned to every signal, so that in the sequel, identical names refer to identical signals. The kernel-language SIGNAL possesses 5 instructions, the first of them being a generic one.

[2] in the common sense, no mathematical definition is referred to here

```
  (i)   R(x1,...,xp)
 (ii)   y := x $1 init x0
(iii)   y := x when b
 (iv)   y := u default v
  (v)   P | Q
```

(i) direct extension of instantaneous relations into relations acting on signals:

$$\mathtt{R(x1,...,xp)} \iff \forall n : R(\mathtt{x1}_n, ..., \mathtt{xp}_n) \text{ holds}$$

where $R(...)$ denotes a relation and the index n enumerates the instants at which the signals `xi` are present. Examples are functions such as `z := x+y` ($\forall n : \mathtt{z}_n = \mathtt{x}_n + \mathtt{y}_n$). A byproduct of this instruction is that *all referred signals must be present simultaneously, i.e. they must have the same clock.* This is a generic instruction, i.e. we assume a family of relations is available. If `R(...)` is the universal relation, i.e., it contains all the p-tuples of the relevant domains, the resulting SIGNAL instruction only constrains the involved signals to have the same clock: the so obtained instruction will be written

```
synchro {x1,...,xp}
```

and only forces the listed signals to have the same clock.

(ii) shift register.

$$\mathtt{y := x\ \$1\ init\ x0} \iff \forall n > 1 : \mathtt{y}_n = \mathtt{x}_{n-1}, \mathtt{y}_1 = \mathtt{x0}$$

Here the index n refers to the values of the signals when they are *present.* Again this instruction forces the input and output signals to have the same clock. In writing actual programs, we shall defer the "`init`" declaration to some specific field of the program.

(iii) condition (`b` is boolean): `y` equals `x` when the signal `x` and the boolean `b` are available and `b` is true; otherwise, `y` is absent; the result is an event-based undersampling of signals. Here follows a diagram summarizing this instruction:

x :	1	2	$\perp$	$\perp$	3	4	$\perp$	$\perp$	5	6	9	...
b :	t	f	t	$\perp$	f	t	f	$\perp$	$\perp$	f	t	...
y :	1	$\perp$	$\perp$	$\perp$	$\perp$	4	$\perp$	$\perp$	$\perp$	$\perp$	9	...

(iv) `y` merges `u` and `v`, with priority to `u` when both signals are simultaneously present; this instruction is the key to oversampling as we shall see later. Here follows a table summarizing this instruction:

u :	1	2	$\perp$	$\perp$	3	4	$\perp$	$\perp$	5	$\perp$	9	...
v :	$\perp$	$\perp$	$\perp$	3	4	10	$\perp$	8	9	2	$\perp$	...
y :	1	2	$\perp$	3	3	4	$\perp$	8	5	2	9	...

Instructions (i-iv) specify the elementary programs.

(v) combination of already defined programs: signals with common names in `P` and `Q` are considered as identical. For example

```
(| y := zy + a
 | zy := y $1 x0 |)
```

denotes the system of recurrent equations:

$$\begin{aligned} y_n &= zy_n + a_n \\ zy_n &= y_{n-1},\ zy_1 = \mathtt{x0} \end{aligned} \tag{14}$$

On the other hand, the program

```
(| y := x when x>0
 | z := y+u |)
```

yields

$$\text{if} \quad x_n > 0 \quad \text{then} \quad \begin{cases} y_n = x_n \\ z_n = y_n + u_n \end{cases} \qquad \text{else} \quad y_n = u_n = z_n = \perp \tag{15}$$

where (x_n) denotes the sequence of present values of `x`. Hence the communication `|` causes $\perp$ to be inserted whenever needed in the second system `z:=y+u`. This is what we wanted for the example (12,13). Let us explain this mechanism more precisely. Denote by $u_1, u_2, u_3, u_4, \cdots$ the sequence of the present values of `u` (recall that `y,z` are present simultaneously with `u`). Then, according to point (1) of the discussion at the beginning of this section, $u_1, u_2, u_3, u_4, \cdots$ is equivalent to its following expanded version:

$$\mathtt{u}\ :\ \ \perp, u_1, \perp, \perp, u_2, u_3, \perp, \perp, \perp, u_4, \perp, \cdots,$$

for any finite amount of "$\perp$"s inserted between successive occurrences of `u`. Assuming all signals of integer type, suppose the following sequence of values is observed for `x`:

$$\mathtt{x}\ :\ \ -2, +1, -6, -4, +3, +8, -21, -7, -2, +5, -9, \cdots$$

Then the amount of inserted "$\perp$"s for the above expanded version of `u` turns out to fit exactly the negative occurrences of `x`: this flexibility in defining `u` allows us to match the present occurrences of `u` with the present occurrences of `y`, i.e., the positive occurrences of `x`.

Structured programming. We provide here some of the features of the actual SIGNAL language for this purpose, for additional details, see [2]. A SIGNAL program is organized as follows :

```
program NAME =
   { ? list_of_inputs
     ! list_of_outputs }          % interface declaration

                                  % begin body of the program
   (| signal_1 := expr_1
    | ...
```

```
        | signal_k := expr_k
        | ...
        | signal_K := expr_K
        |)
                                    % end body of the program

    where                           % declarations
        type of local variables, initializations
    end
```

4.4 An illustrative example : Peterson's algorithm

To illustrate the use of the programming language SIGNAL we present, as an example, a synchronous version of Peterson's algorithm for critical section handling.

Two external devices (*clients*) share a common resource. The two clients obey the following protocol:

1. They send to a controller a request for the resource.
2. They wait until the controller grants them access.
3. When they no longer need it, they inform the controller that the resource becomes free.

Our objective is to design a controller that guarantees correct and fair sharing of the resource : both clients never access the resource together at the same time, and a client, after requesting the resource, will never wait for ever.

The controller. The controller communicates with client i ($i = 1$ or 2) by means of three signals: the input signals `Request_i` and `Free_i` for "request" and "deallocation" of the resource, the output signals `Access_i` grants access to the resource.

The program is the following:

```
process CONTROLLER =
    { ? event Request1, Free1, Request2, Free2
      ! event Access1, Access2 }
```

;The signals Prev_reqi and Zpri memorize the requests

```
    (| Prev_req1 := Request1 default (not Free1) default Zpr1
     | Zpr1 := Prev_req1 $1
     | Prev_req2 := Request2 default (not Free2) default Zpr2
     | Zpr2 := Prev_req2 $1
```

;The signal P and ZP memorize the last owner of the resource

```
     | P := (not Access1) default Access2 default ZP
     | ZP := P $1
```

;Conflict detects simultaneous requests

```
    | Conflict := Request1 when Request2
```

;Access is granted by the controller

```
    | Access1 := (Request1 when not Prev_req2) default (Conflict when ZP)
       default (Free2 when Prev_req1)
    | Access2 := (Request2 when not Prev_req1) default (Conflict when (not ZP))
       default (Free1 when Prev_req2)
```

;The clock of the internal signals Prev_reqi and P is the supremum of the clocks of the inputs

```
    | synchro { Prev_req1, Prev_req2, P,
        Request1 default Request2 default Free1 default Free2 }
    |)
```

;Types declarations

```
where
    logical Prev_req1, Zpr1 init false, Prev_req2, Zpr2 init false,
          P, ZP init true;
    event Conflict
end
```

This program describes the program interface (the {? ..., ! ...} part), the program body and the declaration of local signals. Each signal is typed. A signal of `event` type can be considered as a signal which takes only the value *true*, i.e., it is a clock.

5 Checking properties on SIGNAL programs

5.1 Encoding SIGNAL programs using Galois fields

In this paper, we concentrate on the subset of SIGNAL that involves only `logical` (i.e., boolean) and `event` data types. A corresponding SIGNAL program is an implicit system of multiple clocked dynamical equations which specifies some finite state transition system. Accordingly, the main issues are

- *compiling*, i.e., transforming this implicit system into an equivalent input-output form,
- *checking properties.*

In this paper, we focus on the second aspect. A given signal may have one of the following status : *absent, true, false* as well as the status *present* we consider as a non determinate "true or false" value. The field $\mathcal{F}_3$ of integers modulo 3 is used for this purpose via the following coding :

$$absent \rightarrow 0,\ true \rightarrow +1,\ false \rightarrow -1$$

The relations between clocks and booleans are then encoded as polynomial equations. For example, the expression `a and b = a`, where `a,b` are booleans, is encoded as follows:

$$a^2 \ = \ b^2,\ a^2 = b - a + ab \tag{16}$$

In these equations, the variables a, b refer to infinite sequences of data in $\mathcal{F}_3$ with implicit time index. The first equation of (16) expresses that the two signals `a` and `b` must have the same clock, while the second one encodes the particular boolean relation. Dynamical systems appear when the delay operator ($) is used.

We proceed now on presenting the coding SIGNAL programs.

Boolean Functions. Boolean functions are translated into a system of two equations, one expresses that signals must be synchronous, the other gives the relation between actual values. For example, the program `C := A or B` is encoded as:

$$c^2 = a^2 = b^2$$
$$c = ab(1 - (ab + a + b)).$$

The when instruction. The SIGNAL equation `Y := X when B`, has the polynomial equation $y = x(-b - b^2)$ as coding.

Default If `X`, `Y`, and `Z` are boolean, `Z := X default Y` is translated into: $z = x + (1 - x^2)y$.

The delay (or shift register). The program

```
Y := X $1 init y0
```

is encoded as follows. Introduce a state variable ξ representing the most recent value of `X`, the coding is :

$$\xi' = x + (1 - x^2)\xi \quad (17)$$
$$y = \xi x^2 \quad (18)$$
$$\xi_0 = y_0 \quad (19)$$

where ξ' is the next value of ξ. Equation (18) relates the value of x, the value of y, and the state ξ. Equation (17) describes what will be the following value of the state. A comprehensive coding must include the initial value of ξ corresponding to the initialisation of `Y`. Note that ξ is always present, so it takes only value ± 1.

5.2 Example: the coding of the controller

The polynomial dynamical system encoding this program has the following equations as state equations :

$$\xi_1{}' = pr_1 + (1 - pr_1^2)\xi_1 \ ; \ \xi_{10} = -1$$
$$\xi_2{}' = pr_2 + (1 - pr_2^2)\xi_2 \ ; \ \xi_{20} = -1$$
$$\xi_p{}' = p + (1 - p^2)\xi_p \ ; \ \xi_{p0} = 1$$

The remaining equations are all static equations, i.e., the implicit time index is the same on both sides:

$$h = r_1^2 + (1 - r_1^2)(r_2^2 + (1 - r_2^2)(f_1^2 + (1 - f_1^2)f_2^2))$$

$$
\begin{aligned}
pr_1 &= r_1^2 + (1 - r1^2)(-f_1^2 + (1 - f_1^2)zpr_1) \\
zpr_1 &= h\xi_1 \\
pr_2 &= r_2^2 + (1 - r2^2)(-f_2^2 + (1 - f_2^2)zpr_2) \\
zpr_2 &= h\xi_2 \\
p &= -a_1 + (1 - a_1^2)(a_2 + (1 - a_2^2)zp)) \\
zp &= h\xi_p \\
c &= r_1^2 r_2^2 \\
a_1 &= r_1^2((pr_2 - pr_2^2) + r_2^2(-zp - zp^2)) + \\
&\quad f_1^2(-pr_1 - pr_1^2)(1 - r_1^2((pr_2 - pr_2^2) + r_2^2(-zp - zp^2))) \\
a_2 &= r_2^2((pr_1 - pr_1^2) + r_1^2(zp - zp^2)) + \\
&\quad f_2^2(-pr_2 - pr_2^2)(1 - r_2^2((pr_1 - pr_1^2) + r_1^2(zp - zp^2)))
\end{aligned}
$$

In these equations, the variable h is a shorthand for

`Request1 default Request2 default Free1 default Free2`

The variables r_i, f_i, pr_i, a_i, c are the codings of

`Request_i, Free_i, Prev_req_i, Access_i, Conflict`

respectively.

5.3 Proving some properties of Peterson algorithm

In this section, we apply the toolkit of algorithms we have presented to check various properties on our SIGNAL implementation of Peterson's algorithm.

5.3.1 Safety

The first property we want to prove for our program is the vital condition that *both clients never can access the resource simultaneously.*

In our resource allocation problem, as in most real-time problems, one cannot expect to build a program that works whatever the external world behaves. Hence a description of the interactions between the external world and the program must be included in the specifications. In our example, the controller does not, indeed, work correctly with anarchic clients. It would be confused if some client would send both request and release signals simultaneously. So we will complete the specifications by describing constraints on the behaviour of the clients. The SIGNAL programming language itself can be used for this purpose.

An essential property of the behaviour of clients is the interleaving of requests and releases: `Free_i` (release of the resource by i) occurs exactly once between two subsequent occurences of `Request_i` (request from client i). This is an algebraic property which can be described in SIGNAL using an auxiliary boolean `X_i` for each client:

```
(|  X_i := not ZX_i
 | ZX_i := X_i$1
```

```
| synchro{ Request_i , when X_i}
| synchro{ Free_i,  when ZX_i}
|)
```

where `ZX_i` is initialized to $false$.

The coding of the specifications for each client introduces a new state variable μ_i with associated evolution equation:

$$\mu_i' = x_i + (1 - x_i^2)\mu_i,$$

and a new system of constraints:

$$\begin{aligned} zx_i &= x_i^2\mu_i \\ x_i &= -zx_i \\ r_i^2 &= -x_i - x_i^2 \\ f_i^2 &= -zx_i - zx_i^2. \end{aligned}$$

In order to prove that both clients cannot own the resource simultaneously we need to express the fact "client i is owning the resource". Let us introduce the signals

```
CS1 := Access1 default (not Free1)
CS2 := Access2 default (not Free2)
```

The true (resp. false) values of this signals occur when client i gets (resp. releases) the resource. The definition of this signal is translated into:

$$cs_i = a_i + (1 - a_i^2)l_i.$$

A state variable ν_i holding the last value of cs_i represents exactly what we need: client i owns the resource when ν_i is 1. The evolution equation for ν_i is

$$\nu_i' = cs_i + (1 - cs_i^2)\nu_i.$$

Collecting the polynomial representation of clients, controller, and the equations for cs_i and ν_i, we obtain the overall polynomial dynamical system combining the `CONTROLLER` program with the constraints specified on clients. Safety is now simple to express: the `CONTROLLER` program is safe, for clients having the specified behaviour, if "the set of states V such that $\nu_1 = 1$ and $\nu_2 = 1$ cannot be reached from the initial state".

In order to check this property, we compute the largest invariant subset of the complement of V, and then we check if this largest invariant subset contains the initial state. Let us proceed on using our toolkit.

The ideal associated to the complement of V in $\mathcal{F}_3^n$ is generated by the polynomial :

$$\nu_1\nu_2(1 + \nu_1)(1 + \nu_2)$$

completed by all polynomials of the form $\xi^3 - \xi$ where ξ is a state variable. Recall that our state variables never take the zero value, cf. Subsection 5.1. As a consequence, we can restrict our set to the complement of V in $(\mathcal{F}_3 - \{0\})^n$, by adding the generators $\xi^2 - 1$ for all state variable ξ.

Hence, the complement of V in $(\mathcal{F}_3 - \{0\})^n$ is defined by the ideal:

$$\begin{aligned} I &= < \nu_1\nu_2(1+\nu_1)(1+\nu_2), \\ &\quad \nu_1^2 - 1,\ \nu_2^2 - 1,\ \mu_1^2 - 1,\ \mu_2^2 - 1,\ \xi_1^2 - 1,\ \xi_2^2 - 1,\ \xi_p^2 - 1 > . \end{aligned}$$

The algorithm that computes the largest invariant subset of $\mathcal{V}(I)$ yields:

$$\begin{aligned} J &= < (1+\nu_1)(1+\nu_2),\ \nu_1^2 - 1,\ \nu_2^2 - 1, \\ &\quad \mu_1^2 - 1,\ \mu_2^2 - 1,\ \xi_1^2 - 1,\ \xi_2^2 - 1,\ \xi_p^2 - 1, \\ &\quad (1+\nu_1)(1-\xi_1),\ (1+\nu_2)(1-\xi_2), \\ &\quad (1+\xi_1)(1-\mu_1),\ (1+\xi_2)(1-\mu_2), \\ &\quad (1+\nu_1)(1-\mu_1),\ (1+\nu_2)(1-\mu_2) > . \end{aligned}$$

as the characteristic ideal of the largest invariant subset of $\mathcal{V}(I)$. It is easy to check that the initial state defined by: $\nu_1 = \nu_2 = \xi_1 = \xi_2 = \mu_1 = \mu_2 = -1$ and $\xi_p = 1$ belongs to $\mathcal{V}(J)$.

5.3.2 Liveness

Liveness of the controller is the property that "at least one client can own the resource". This property can be established by proving that "the set of states where one of the clients is owning the resource" is reachable from the initial state. Denote by L the so defined set of states. The ideal defining L is generated by the following polynomial:

$$(\nu_1 - 1)(\nu_2 - 1).$$

The method is similar to the previous one with the modification that we expect the initial state not to belong to the largest invariant subset of the complement of L.

5.3.3 Fairness

The fairness of the controller can be proved by checking the absence of starvation: "a client after asking for the resource, will eventually get it".

Let us consider client 1. Extract the following relations from the specifications of the controller (first two instructions) and the SIGNAL description of the behaviour of the clients (last instruction) :

```
(|  Prev_req1 := Request1 default (not Free1) default Zpr1
 | Zpr1 := Prev_req1$1
 | CS1 := Request1 default (not Free1)
 |)
```

Client 1 is waiting from the instant when `Request1` occurs until `Access1` or `Free1` occurs. Consider the state variables ξ_1 and ν_1 that respectively memorize the value of `Prev_req1` and `CS_1`. The states in which client 1 is in the status "waiting" satisfy the equations :

$$\xi_1 = 1\ ,\ \nu_1 = -1.$$

Let us call W_1 the set of such states. Starvation of client 1 occurs if there exists an infinite trajectory of the state included in W_1. Thus starvation can not occur if and only if "W_1 contains no stabilisable subset". This is checked by showing that the largest stabilisable subset of W_1 is empty. Let us proceed on checking this last property.

For a polynomial system obtained from a SIGNAL program, the t-uple of signals $y = (0, \ldots, 0) \in \mathcal{F}_3^m$ is admissible at every state x and causes no transition to occur, i.e., is such that $P(x, y) = x$. Hence, for such a system, any set of state is stabilizable. But this "zero" t-uple of signals corresponds to the absence of activity of the program. Performing nothing is always allowed for a SIGNAL program, but is of course of little interest. So, prior to study stabilization, we have to remove this "zero" signal by adding the equation

$$\prod_{i=1}^{m}(1 - y_i^2) = 0 \tag{20}$$

to the set of constraints.[3]

As previously noticed, we only have to consider non-nul states. W_1 is then described by the ideal:

$$\mathcal{I}(W_1) =< \xi_1 - 1, \nu_1 + 1, \xi_2^2 - 1, \xi_p^2 - 1, \nu_2^2 - 1, \mu_1^2 - 1, \mu_2^2 - 1 > .$$

The largest stabilisable subset of W_1 is then computed and happens to be empty. Obviously, the same property holds when client 2 is considered.

6 Controlled dynamical systems

In the Ramadge and Wonham theory Discrete Event Systems, the physical system (the plant) is modelled by means of a DES (most often specified by a finite state automata). The control of the plant is performed by inhibiting some events belonging to a set of controlled events while the other events cannot be prevented to occur. It appears that this point of view although general is not always the most convenient from a practical point of view.

In some situations the relations between the plant and the controller are best described by considering the plant as emitting signals to the controller which in turn emits signals to control the plant. In this paper we will make a distinction between the two situations. Even if it is possible to mix the two approach in a single model, we feel that the two approaches are complementary in applications.

6.1 Systems controlled by signals

A dynamical system controlled by means of signals is defined by three set of equations:

$$\left\{ \begin{array}{rcl} X' & = & P(X, Y, U) \\ Q(X, Y, U) & = & 0 \\ Q_0(X_0) & = & 0 \end{array} \right.$$

where X has the same meaning as above but now we distinguish between uncontrolled signals Y and controlled ones U. That is to say: we can at each instant n, given x_n and y_n, chose some

[3] It is not necessary to remove "zero" signals to check safety properties, since the additional constraint (20) does not modify invariance properties.

u_n which is admissible i.e. such that $Q(x_n, y_n, u_n) = 0$. Such systems will be termed systems controlled by signals or signal-controlled systems.

If we consider a value of a signal as an event (which implies that some value may occur on a signal while no value occurs on an another one), this model allows simultaneous events.

An important property of dynamical systems controlled with signals is control-invariance. The definition is adapted from the preceding one:

Definition 5 *A set of states E is control-invariant for a controlled dynamical system if for each state x in E and each admissible y, there exists a signal value u such that $P(x, y, u)$ is in E.*

This property is easily translated for ideals:

Proposition 6 *A set of states E is control invariant for a dynamical system controlled with signals if and only if*

$$(\underline{c} + \ < P^*(\mathcal{I}(E)) >) \cap \mathcal{F}_p[X, Y] \subseteq \mathcal{I}(E)\mathcal{F}_p[X, Y] \ + \underline{c}_{xy}$$

where $\underline{c}_{xy} = \underline{c} \cap \mathcal{F}_p[X, Y] = < Q(X, Y) > \cap \mathcal{F}_p[X, Y]$.

6.2 Systems controlled by event inhibitions

To introduce the model of dynamical systems controlled by means of inhibiting events let us consider general dynamical systems:

$$\left\{ \begin{array}{rcl} X' & = & P(X, Y) \\ Q(X, Y) & = & 0 \\ Q_0(X_0) & = & 0 \end{array} \right.$$

Now we consider a value y of Y as an event. The set of controllable events is then defined by the associated ideal $< Q_c(Y) >$ and the set of uncontrollable events has associated ideal $< Q_u(Y) >$. These two sets must induce a partition of the set of events, thus the following relations must be satisfied:

$$\begin{array}{rcl} < Q_c(Y)Q_u(Y) > & \subseteq & Q(X, Y) \cap \mathcal{F}_p[Y] \\ < Q_c(Y), Q_u(Y) > & = & < 1 > \end{array}$$

Such systems will be called systems controlled by events or event-controlled systems.

Various definitions from Radmage and Wonham theory can be adapted to the dynamical systems over Galois fields. We need in particular the following:

Definition 6 *A state feedback for an event-controlled dynamical system is an ideal $< F(X, Y) >$ in $\mathcal{F}_p[X, Y]$ such that:*

$$< F(X, Y) > \subseteq < Q_u > \mathcal{F}_p[X, Y] + \ < Q >$$

This definition simply states that the feedback admits any uncontrollable event which is admissible by the plant. The control-invariant property is also adapted from the linguistic formulation:

Definition 7 *A set of states E is control-invariant if and only if there exists a state feedback $< F(X, Y) >$ such that E is invariant for the dynamical system:*

$$\left\{ \begin{array}{rcl} X' & = & P(X, Y) \\ Q(X, Y) & = & 0 \\ F(X, Y) & = & 0 \\ Q_0(X_0) & = & 0 \end{array} \right.$$

The following proposition is a straightforward translation of the equivalent proposition in [14].

Proposition 7 *A set E is control-invariant if and only if it is invariant for the dynamical system:*

$$\begin{cases} X' &= P(X,Y) \\ Q(X,Y) &= 0 \\ Q_i(Y) &= 0 \\ Q_0(X_0) &= 0 \end{cases}$$

An admissible feedback controlling the system such as E is invariant is $F(X,Y) = Q_i(Y)$. Of course this feedback is the most restrictive for this purpose. It may even be too restrictive: if in some state $x \in E$ the only admissible events are the controllable ones, x would not be alive for the controlled system. Too much restrictive controllers may lead to deadlock situations. This simple example shows the need for a definition of *admissible controllers*.

6.3 Properties of reactive systems

In traditional control theory, various control problems are considered but the starting point is in general: given a model of the plant (based on differential equations or on difference equations) and a *control objective*, a controller is derived by various means such that the closed loop system behaviour meets the control objective. Growing experience has allowed for the emergence of some fundamental control objectives: pole placement, disturbance rejection, decoupling ....

In the area of DES, not only the models of the plants are very diverse (linguistic models, Petri nets, automata, max-plus algebra, polynomial dynamical systems ....) but the basic control objectives are not yet clearly identified and classified. In the Ramadge and Wonham approach, the control objectives are given in terms of languages. Although any objective can be specified that way, this kind of specification is not of common use in the control community.

DES belongs to the family of reactive systems, that are systems continuously interacting with their environment. These systems have been studied in the area of computer science where they appear as models for protocols, real-time systems, computer networks .... Some basic properties of these systems have been isolated and studied. We propose to ground our control objective on these fundamental properties as exposed in the synthetic paper [15]. This paper deals with the properties of sequences of events taken from a finite set Σ.

The properties of finite sequences are defined by a language Φ. In most applications this language is easy to specify (much more easy than an infinitery language). Given a sequence $s \in \Sigma^\omega$ let us denote by $pref_n(s)$ the prefix of s: $s_0 s_1 \ldots s_n$.

Following [15] we define several sets of sequences or $\omega-$languages:

- $A(\Phi)$ is the set of sequences s in Σ^ω such that for all n, $pref_n(s)$ is a word in Φ.
- $E(\Phi)$ is the set of sequences s in Σ^ω such that there exists n and $pref_n(s)$ is in Φ.
- $R(\Phi)$ is the set of sequences s in Σ^ω such that $pref_n(s)$ is a word in Φ for infinitely many n.
- $P(\Phi)$ is the set of sequences s in Σ^ω such that $pref_n(s)$ is in Φ for all n but a finite number.

A property on sequences is then defined as the set of all the sequences having that property. Following Manna and Pnueli we define:

- $A(\Phi)$ as the *safety* property induced by Φ.
- $E(\Phi)$ as the *guarantee* property induced by Φ.
- $R(\Phi)$ as the *recurrence* property induced by Φ.
- $P(\Phi)$ as the *persistence* property induced by Φ.

Let us denote by $\mathcal{A}$ the set of all languages defining safety properties, $\mathcal{E}$, $\mathcal{R}$,$\mathcal{P}$ the sets of languages defining guarantee, recurrence and persistence respectively. Relationships exist between these different classes of languages. In particular $\mathcal{A}$ and $\mathcal{E}$ are dual classes for the complementation of sets as are $\mathcal{R}$ and $\mathcal{P}$.

Topological properties of these classes have been studied. In particular, safety properties correspond to closed subsets in Σ^ω for the product topology. The relations of these classes with temporal logic formulas make them interesting for specifying infinitary properties of reactive systems.

6.4 Control: State properties

We will consider at first very simple properties on states. These properties are defined by a set of states or equivalently by an ideal $\underline{a} \subseteq \mathcal{F}_p[X]$ defining a set of states. We may consider that $\mathcal{V}(\underline{a})$ is the set of "good states" and thus a safety property associated with $\mathcal{V}(\underline{a})$ simply says: $\forall n\ x_n \in \mathcal{V}(\underline{a})$. This condition implies $x_0 \in \mathcal{V}(\underline{a})$, so a controller must take care of the initial states. All the properties which are to be considered in this section can be settled using only *one* time index. For example safety:$\forall n\ x_n \in \mathcal{V}(\underline{a})$ or attractivity $\exists n\ x_n \in \mathcal{V}(\underline{a})$ are specified with relations involving only one time index. For that reason we will call them static or order zero properties.

6.4.1 Admissible state feedbacks

Admissible state feedbacks have different definitions depending whether the dynamical system is signal-controlled or event-controlled.

Definition 8 *A state feedback for a signal-controlled dynamical system is a couple of equations (or ideals)* $(C_0(X),\ C(X,Y,U))$. *The controlled system is the dynamical system obtained by adding the feedback equations to the initial system. This feedback is admissible for the signal-controlled dynamical system (S):*

$$\left\{\begin{array}{rcl} X' & = & P(X,Y,U) \\ Q(X,Y,U) & = & 0 \\ Q_0(X_0) & = & 0 \end{array}\right.$$

if it satisfies the following conditions:

- *The initial constraints C_0 and Q_0 have common roots.*
- *For all state x in the orbit of the controlled system, every event y which is admissible at x for the initial system is also admissible for the controlled system.*

The last condition indicates that the signals y cannot be controlled. The control equation $C(X,Y,U)$ may be read as: given a state x_n and an uncontrolled signal y_n, chosing a u_n such that $C(x_n, y_n, z_n) = 0$ implies an evolution of the state in accordance with the control objective.

The proof of the following proposition is straightforward:

Proposition 8 *Given a signal-controlled dynamical system (S) and a set of states E, there exists an admissible state feedback such that E is invariant for the controlled system if and only if:*

- $< Q_0(X) > + \mathcal{I}(E) \neq 1$
- *E is control-invariant for the dynamical system (S).*

An admissible feedback is then: $< C_0(X) > = \mathcal{I}(E)$, $C(X,Y,U) = P^*(\mathcal{I}(E))$.

Event-controlled systems As for signal-controlled dynamical systems, a controller for an event-controlled dynamical system is a pair (C_0, C) where C_0 is a constraint on the initial state which must be compatible with the initial constraints of the system. The feedback part $C(X,Y)$ must be a feedback for event-controlled dynamical systems. Moreover we impose the liveness property to the controlled system.

6.4.2 Some control problems

The properties of sequences will be used for the definition of the control objectives. Since static state properties are under consideration, the finite language Φ is easy to define: a finite sequence $x_0 x_1 \ldots x_k$ is in Φ if and only if x_k is in the set of "good states" E. So the associated safety property states: $\forall n \ x_n \in E$ i.e. E is an invariant set and the initial state is in E. Similarly, the guarantee property translates into: E is a reachable state, the recurrent property into: E is a recurrent set and the persistence property into: E is invariant and attractive.

We will consider control objectives which are conjunction of basic properties of state trajectories. It must be mentioned that basic properties cannot be combined, in general, in a modular way. For example, a safety property put restrictions on the set of state trajectories which may be not compatible with a guarantee property. The synthesis of a controller insuring both properties must be done by considering both properties *simultaneously* and not by combining a controller insuring safety with a controller insuring guarantee independantly.

6.4.3 Safety

A dynamical system satisfies a state safety property if for all n the state x_n belongs to the set of "good states" E. Let us first consider signal-controlled systems. The controller is correct if it is acceptable and the orbit of the controlled system

$$\left\{ \begin{array}{rcl} X' & = & P(X,Y,U) \\ Q(X,Y,U) & = & 0 \\ C(X,Y,U) & = & 0 \\ C_0(X_0) & = & 0 \end{array} \right.$$

is included in E. This is equivalent to: there exists a control-invariant subset of E containing the initial states. If the problem is solved, the orbit of the controlled system has this property. On the other hand, suppose a control-invariant subset O of E is given, then it is easy to prove that an acceptable controller is:

$$\left\{ \begin{array}{rcl} < C(X,Y,U) > & = & < P^*(\mathcal{I}(O))(X,Y,U) > \\ < C_0(X_0) > & = & \mathcal{I}(O) \end{array} \right.$$

The first equation insures that the next state will be in O while the second equation forces the initial state to be in O. Thus solving the control problem reduces to find a control-invariant subset of E. This is done by computing the greatest control-invariant subset of E. The algorithm is derived from (5):

$$\begin{aligned}
\underline{a}_0 &= \mathcal{I}(E) \\
\underline{a}_1 &= \underline{a}_0 + coef_X(c^c_{X,Y} \cap [\mathcal{F}_p[X,Y] \cap (< Q > + < P^*(\underline{a}_0) >)]) \\
&\dots \\
\underline{a}_{k+1} &= \underline{a}_k + coef_X(c^c_{X,Y} \cap [\mathcal{F}_p[X,Y] \cap (< Q > + < P^*(\underline{a}_k) >)])
\end{aligned}$$

Let us introduce the operator $Isc(\underline{a}) = coef_X(c^c_{X,Y} \cap [\mathcal{F}_p[X,Y] \cap (< Q > + < P^*(\underline{a}) >)])$. The algorithm may be rewritten now:

$$\begin{aligned}
\underline{a}_0 &= \mathcal{I}(O) \\
\underline{a}_{k+1} &= \underline{a}_k + Isc(\underline{a}_k)
\end{aligned}$$

The increasing sequence of ideals is eventually stationnary. The limit is either the entire ring and the control problem has no solution, either a proper ideal corresponding to the greatest control-invariant subset of E.

Event-driven systems If a controller exists insuring the state safety property defined by E then the orbit O of the controlled system:

$$\left\{ \begin{array}{rcl} X' &=& P(X,Y) \\ Q(X,Y) &=& 0 \\ C(X,Y) &=& 0 \\ C_0(X_0) &=& 0 \end{array} \right.$$

is a subset of E which is invariant with respect to uncontrollable events (the controller always accept them) and since the controller is non-blocking the orbit is also control-invariant for the original dynamical system. The set O satisfies both properties:

$$\left\{ \begin{array}{rcl} coef_x(P^*(\mathcal{I}(O)) \cap (\underline{c} + \underline{e}_i)^c) &\subseteq& \mathcal{I}(O) \\ (\underline{c} + P^*(\mathcal{I}(O)) \cap \mathcal{F}_p[X] &\subseteq& \mathcal{I}(O) \end{array} \right.$$

Since the two properties are invariant under set unions, the conjunction of them is also invariant under set unions. As a consequence, there exists a greater subset of E with both properties. We have already introduced algorithms computing the greatest subsets for each property. The algorithm for the conjunction of the two properties interleaves the two algorithms as follow:

$$\begin{aligned}
\underline{a}_0 &= \mathcal{I}(E) \\
\underline{a}_{2k+1} &= \underline{a}_{2k} + coef_X(\underline{a}_{2k} \cap (\underline{c} + \underline{e}_i)^c) \\
\underline{a}_{2k+2} &= \underline{a}_{2k+1} + (\underline{c} + P^*(\underline{a}_{2k+1})) \cap \mathcal{F}_p[X]
\end{aligned}$$

If the limit is not the entire ring, the controller is obtained as in the preceding case.

6.4.4 Safety+reachability

Let us denote E_s the set of states representing the safety property and E_r the set of states representing the reachability property. The reachability we have in mind is the following: the controller allows the state of the controlled dynamical system to reach E_r. It doesn't mean that it is possible to control the system to enforce this property. The controller we are looking for insures that all the trajectories of the controlled system are included in E_s while for each point of the orbit there exists a safe trajectory starting in that point and reaching E_r.

Signal-controlled systems For all set of states E and for all dynamical system, let us denote $Att(E, E_r)$ the set of states x such that: there exists a trajectory of the dynamical system, starting in x, included in E and reaching E_r.

Should an acceptable controller insuring the safety and the reachability property exists, then the orbit O of the controlled system would have the properties:

- $O \subseteq E_s$
- O is control-invariant (for the original system)
- $O \subseteq Att(O, E_r)$

Reciproquely, if a subset E of E_s has the preceding properties, it is not difficult to show that a satisfactory controller is obtained as previously, provide the initial constraints are compatible.

Let us remark that the property $E = Att(E, E_r)$ is also invariant under set unions as is the control-invariance. Thus the problem of finding a controller insuring both safety and reachability reduce to the computation of the greatest subset of E_s such that:

- E is control invariant
- $E = Att(E, E_r)$

The problem will be completely solved by providing an algorithm to compute the ideal associated with $Att(E, E_r)$. Let us introduce the function pre:

$$pre(\underline{a}) = \mathcal{F}_p[X] \cap (P^*(\underline{a}) + < Q >)$$

$\mathcal{V}(pre(\mathcal{I}(E)))$ is the set of states having at least a successor in E. $\mathcal{I}(Att(E, E_r))$ is then the limit of the sequence of ideals:

$$\begin{aligned} \underline{a}_0 &= \mathcal{I}(E) + \mathcal{I}(E_r) \\ \underline{a}_{k+1} &= \underline{a}_k.(\mathcal{I}(E) + pre(\underline{a}_k)) \end{aligned}$$

The dot denotes the product of ideals corresponding to the union of the corresponding sets. This sequence is a decreasing sequence of completed ideals, so it is stationnary.

The greatest subset of E_s having the two properties is then computed by the following algorithm:

$$\begin{aligned} \underline{a}_0 &= \mathcal{I}(E_s) \\ \underline{a}_{k+1} &= Att(\underline{a}_k + Isc(\underline{a}_k), E_r) \end{aligned}$$

mixing the algorithm for the greatest control-invariant with the algorithm for obtaining the greatest subset such that $E = Att(E, E_r)$, (the function $Att(\ ,\ )$ on ideals corresponding to the function on sets bearing the same name).

Event-controlled systems The solution of the control problem for event-controlled systems can be deduced from the one obtained for signal-controlled systems. If the control problem has a solution, then the orbit O of the controlled system must have the following properties:

- $O \subseteq E_s$,
- O is control-invariant for the original system,
- O is invariant with respect to uncontrollable events,
- $O \subseteq Att(O, E_r)$.

The last three properties are stable under set unions. The solution is obtained as above by computing the greatest subset of E_s having the three properties.

6.4.5 Safety+Guarantee

This control objective is somewhat a dual of the preceding one. In addition to safety, it is asked that every trajectory of the controlled system *must* reach a given set of states E_r.

Signal-controlled systems We consider again a signal-controlled system and suppose that a solution to the control problem exists. The orbit O of the controlled system has then two properties:

- O is included in E_s,
- for all state x in O, any trajectory starting at x reaches E_r.

Let us define now for a set of states E, $\widetilde{Att}(E, E_r)$ as the set of states x such that every trajectory initialized in x and entirely included in E reaches E_r. A necessary and sufficient condition for the existence of a solution to the control problem is then the existence of a set E such that:

- $E \subseteq E_s$,
- E is control invariant for the original system,
- $E = \widetilde{Att}(E, E_r)$
- $E \cap \mathcal{V}(Q_0) \neq \emptyset$

As in the preceding case, the conjunction of the property $E = \widetilde{Att}(E, E_r)$ and the control invariance is stable under set union. Consequently there exists a greater subset of E_s with the two properties. A controller can be derived from the ideal associated with this set if the last condition is satisfied.

It remains to effectively compute this set. A necessary stage in this computation is the computation of $\widetilde{Att}(E, E_r)$. Let us introduce the function $\widetilde{pre}$:

$$\widetilde{pre}(\underline{a}, \underline{b}) = coef_X(P^*(\underline{a}).(\underline{c} + P^*(\underline{b}))^c)$$

The set $\mathcal{V}(\widetilde{pre}(\underline{a}, \underline{b}))$ is the set of states x such that for all admissible y, if $P(x, y)$ is in $\mathcal{V}(\underline{b})$ then $P(x, y)$ is in $\mathcal{V}(\underline{a})$. That is to say: every successor of x belonging to $\mathcal{V}(\underline{b})$ is in $\mathcal{V}(\underline{a})$. The computation

of the ideal associated with $\widetilde{Att}(E, E_r)$ follows a path similar to the computation of Att. It is the limit of the decreasing sequence of completed ideals:

$$\begin{aligned} \underline{a}_0 &= \mathcal{I}(E) + \mathcal{I}(E_r) \\ \underline{a}_{k+1} &= \underline{a}_k.(\mathcal{I}(E) + \widetilde{pre}(\underline{a}_k, \mathcal{I}(E))) \end{aligned}$$

The computation of the greatest subset of E_s with the desired properties is similar to the computation in the preceding case. The corresponding ideal is the limit of the increasing chain of ideals:

$$\begin{aligned} \underline{a}_0 &= \mathcal{I}(O_s) \\ \underline{a}_{k+1} &= \widetilde{Att}(\underline{a}_k, \mathcal{I}(O_2)) \\ \underline{a}_{k+2} &= \underline{a}_{k+1} + Isc(\underline{a}_{k+1}) \end{aligned}$$

The existence of a controller is then obtained by checking the compatibility of the initial conditions.

Event-controlled systems The solution is very similar to the one obtained in the case of the safety + reachability property. The controller can be computed by simply replacing $Att(E, E_r)$ with $\widetilde{Att}(E, E_r)$.

6.4.6 Safety + persistence

Our last example of control objective specified as a static state property will be the conjunction of a safety property and of a persistence property. The persistence property defined by the set of states E_p, implies not only the attractivity of E_p as in the preceding case but also it's invariance for the controlled system. So the existence of a controller implies the control-invariance of the set E_p. If E_p is not control-invariant, it is necessary to replace E_p with its greatest control-invariant subset. Of course, if this subset is empty, the control problem has no solution. From now, we shall consider E_p as a control-invariant subset of states.

Let g be a generator of the ideal obtained by computing a controller insuring the safety property and the attractivity of E_p and g_p a generator of $\mathcal{I}(E_p)$. The two controllers:

$$C_1(X, Y, U) = P^*(g);\ C_{10}(X) = g(X)$$

insuring safety and attractivity of E_p and

$$C_2(X, Y, U) = (1 - g_p)P^*(g_p)$$

insuring the invariance of E_p do not interact. The solution of the control problem is then obtained by putting the two controllers together.

6.5 Control: Locally testable properties

Many properties of discrete event system cannot be stated with the help of static relations. For example if we don't want a signal y never to take the same value two consecutive times, it must satisfy the relation: $\forall n\ y_{n+1} - y_n \neq 0$. Such relations are sometime called dynamic relations or

relations of order k when they involve time indexes from n to $n+k$. Such relations may also be used for defining implicit dynamical systems.

A locally testable property is a property of trajectories which can be tested locally i.e. on finite segments of trajectories. Given a polynomial $p(y_0, \ldots, y_k)$, we consider the finite language $\Phi = \{y_0 y_1 \ldots y_n / \ n \geq k \ \& \ p(y_{n-k}, \ldots, y_n) = 0\}$. The safety property derived from this language states: $\forall n \ p(y_n, \ldots, y_{n+k}) = 0$. In the same way, the guarantee property derived from Φ is: $\exists n \ p(y_n, \ldots, y_{n+k}) = 0$. The recurrent and persistent properties are obtained similarly.

The solution of control problems involving dynamical locally testable properties uses a technique which is classical in control theory: we simply augment the state of the dynamical system. Given a dynamical system (P, Q) and a locally testable property defined with a polynomial: $P_p(X_n, \ldots, X_{n-k}, Y_{n-1}, \ldots, Y_{n-k})$, we define a new dynamical system (P_a, Q_a) with state:

$$Z_n \ = \ (X_n, \ldots, X_{n-k}, Y_{n-1}, \ldots, Y_{n-k})$$

and events T_n, U_n. The evolution equation is obtained by setting $Y_n = T_n$, $X_{n+1} = P(X_n, Y_n, U_n)$ and shifting the other components. The constraint equation is simply: $Q_a(Z_n, T_n, U_n) = Q(X_n, Y_n, U_n)$.

The dynamical property for the original system is then a static state property for the new system and the control problem is solved with the help of the techniques of the previous section.

6.6 Control: other properties

It is a well known fact in language theory that the locally testable languages form a proper subset of the class of regular languages [16]. A property defined by means of a regular language and not locally testable is in general specified with an automaton or in our context with a dynamical system. There is no room in this paper to develop the computational techniques for this type of control specifications, but the basic idea is to combine the two dynamical systems into one and transform the control specification into a state property.

7 Conclusion

We have presented applications of the theory of dynamical systems over Galois fields to the verification and the synthesis of logic Discrete Events Systems. The example of the SIGNAL programming language gives some clues on how a programming environment for DES could be organized. For many reasons, engineers prefer to program their own controllers. So they need programming facilities including a checker. Nevertheless, in some complex cases, automatic synthesis is necessary. Moreover, in case of error, they need also a diagnosis facility which may use both aspects: checking some properties and synthesis of bad behaviours of the system to give some insight on where the errors are. For all that purposes, dynamical systems over Galois fields give the theorical basis together with efficient algorithms.

References

[1] A. Benveniste, G.Berry. *The Synchronous Approach to Reactive and Real-Time Systems*, Proc. of the IEEE, vol 79, no 9, september 1991,1270-1282.

[2] P. Le Guernic, T. Gauthier, M. Le Borgne, and C. Lemaire. *Programming Real-Time Applications with* SIGNAL , Proc. of the IEEE, vol 79, no 9, september 1991,1321-1336.

[3] F. Boussinot and R. de Simone *The* ESTEREL *Language,* Proc. of the IEEE, vol 79, no 9, september 1991,1293-1304.

[4] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud *The Synchronous Data Flow Programming Language* LUSTRE Proc. of the IEEE, vol 79, no 9, september 1991,1305-1320.

[5] M. Le Borgne, A. Benveniste, P. Le Guernic. *Polynomial ideal theoretic methods in Discrete Event and Hybrid Dynamical Systems*, Proceeding of the 28th Conference on Decision and Control, IEEE Control Systems Society, Volume 3 of 3, 1989, 2695-2700.

[6] M. Le Borgne, A. Benveniste, P. Le Guernic. *Polynomial Dynamical Systems over Finite Fields*,In First European Conference on Algebraic Computing in Control, Paris, march 91.

[7] B. Dutertre *Spécification et preuve de systèmes dynamiques: Application à* SIGNAL . PhD Thesis, University of Rennes, 1992.

[8] A. Pnueli. *The Temporal Logic of Programs.* IEEE Symposium on the Foundations of Computer Science, Providence, Rhode Island, 1977.

[9] E.M. Clarke, P.A. Sistla and E.A. Emerson. *Automatic Verification of Finite State Concurrent Systems using Temporal Logic*. In Symp. on Principles of programming languages, A.C.M., 1983.

[10] M. Vardi. *Verification of Concurrent Programs: the Automata-Theoretic Framework.* In Symp. on Logic in Computer Science, IEEE 1987, 167-176.

[11] R.E.Bryant. *Graph-Based Algorithms for Boolean Function Manipulations.* IEEE Transaction on Computers, vol C-35, no 8, aug 1986, 677-691.

[12] O. Coudert, C. Berthet, J. C. Madre. *Verification of Synchronous Sequencial Machines Based on Symbolic Execution*, in Lecture Notes in Computer Sciences, vol 407, J. Sifakis Editor, Springer-Verlag, june 1989, 365-373.

[13] O. Coudert, C. Berthet, J. C. Madre. *Verifying Temporal Properties of Sequential Machines Without Building their State Diagram*, in Lecture Notes in Computer Sciences, vol 531, CAV'90, E. M. Clarke and R. P. Kurshan Editors, June 1990, 23-32.

[14] P.J. Ramadge and W.M. Wonham *Modular Feedback Logic for Discrete Events Systems* Siam J. Control and Optimisation, vol 25,no 5,sept 1987

[15] Z. Manna and A. Pnueli *A hierarchy of Temporal Properties* Tech. Report, Weizman Institute, Israel.

[16] R. McNaughton and S. Papert *Counter-Free Automata* The M.I.T. Press, 1971.